



HOWTO build a simple RT application

The POSIX [API](#) forms the basis of real-time applications running under PREEMPT_RT. For the real-time thread a POSIX thread is used (pthread). Every real-time application needs proper handling in several basic areas like scheduling, priority, memory locking and stack prefaulting.

Basic prerequisites

Three basic prerequisites are introduced in the next subsections, followed by a short example illustrating those aspects.

Scheduling and priority

The [scheduling policy](#) as well as the priority must be set by the application explicitly. There are two possibilities for this:

1. Using `sched_setscheduler()`

This function needs to be called in the start routine of the pthread before calculating RT specific stuff.

2. Using pthread attributes

The functions `pthread_attr_setschedpolicy()` and `pthread_attr_setschedparam()` offer the interfaces to set policy and priority. Furthermore scheduler inheritance needs to be set properly to `PTHREAD_EXPLICIT_SCHED` by using `pthread_attr_setinheritsched()`. This forces the new thread to use the policy and priority specified by the pthread attributes and not to use the inherit scheduling of the thread which created the real-time thread.

Memory locking

[See here](#)

Stack for RT thread

[See here](#)

Example

```
/*
 * POSIX Real Time Example
 * using a single pthread as RT thread
 */

#include <limits.h>
#include <pthread.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>

void *thread_func(void *data)
{
    /* Do RT specific stuff here */
    return NULL;
}

int main(int argc, char* argv[])
{
    struct sched_param param;
    pthread_attr_t attr;
    pthread_t thread;
```

```

int ret;

/* Lock memory */
if(mlockall(MCL_CURRENT|MCL_FUTURE) == -1) {
    printf("mlockall failed: %m\n");
    exit(-2);
}

/* Initialize pthread attributes (default values) */
ret = pthread_attr_init(&attr);
if (ret) {
    printf("init pthread attributes failed\n");
    goto out;
}

/* Set a specific stack size */
ret = pthread_attr_setstacksize(&attr, PTHREAD_STACK_MIN);
if (ret) {
    printf("pthread setstacksize failed\n");
    goto out;
}

/* Set scheduler policy and priority of pthread */
ret = pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
if (ret) {
    printf("pthread setschedpolicy failed\n");
    goto out;
}
param.sched_priority = 80;
ret = pthread_attr_setschedparam(&attr, &param);
if (ret) {
    printf("pthread setschedparam failed\n");
    goto out;
}
/* Use scheduling parameters of attr */
ret = pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
if (ret) {
    printf("pthread setinheritsched failed\n");
    goto out;
}

/* Create a pthread with specified attributes */
ret = pthread_create(&thread, &attr, thread_func, NULL);
if (ret) {
    printf("create pthread failed\n");
    goto out;
}

/* Join the thread and wait until it is done */
ret = pthread_join(thread, NULL);
if (ret)
    printf("join pthread failed: %m\n");

out:
return ret;
}

```